PotLLL : a polynomial time version of LLL with deep insertions

Fontein, Felix ; Schneider, Michael ; Wagner, Urs

Abstract: Lattice reduction algorithms have numerous applications in number theory, algebra, as well as in cryptanalysis. The most famous algorithm for lattice reduction is the LLL algorithm. In polynomial time it computes a reduced basis with provable output quality. One early improvement of the LLL algorithm was LLL with deep insertions (DeepLLL). The output of this version of LLL has higher quality in practice but the running time seems to explode. Weaker variants of DeepLLL, where the insertions are restricted to blocks, behave nicely in practice concerning the running time. However no proof of polynomial running time is known. In this paper PotLLL, a new variant of DeepLLL with provably polynomial running time, is presented. We compare the practical behavior of the new algorithm to classical LLL, BKZ as well as blockwise variants of DeepLLL regarding both the output quality and running time.

DOI: https://doi.org/10.1007/s10623-014-9918-8

Des. Codes Cryptogr. (2014) 73:355–368
DOI 10.1007/s10623-014-9918-8

PotLLL: a polynomial time version of LLL with deep insertions

Felix Fontein · Michael Schneider · Urs Wagner

Received: 29 July 2013 / Revised: 24 December 2013 / Accepted: 1 January 2014 /
Published online: 22 January 2014

Abstract Lattice reduction algorithms have numerous applications in number theory, algebra, as well as in cryptanalysis. The most famous algorithm for lattice reduction is the LLL algorithm. In polynomial time it computes a reduced basis with provable output quality. One early improvement of the LLL algorithm was LLL with deep insertions (DeepLLL). The output of this version of LLL has higher quality in practice but the running time seems to explode. Weaker variants of DeepLLL, where the insertions are restricted to blocks, behave nicely in practice concerning the running time. However no proof of polynomial running time is known. In this paper PotLLL, a new variant of DeepLLL with provably polynomial running time, is presented. We compare the practical behavior of the new algorithm to classical LLL, BKZ as well as blockwise variants of DeepLLL regarding both the output quality and running time.

Keywords Lattice reduction · LLL algorithm · Deep insertion

Mathematics Subject Classification 68R05 · 94A60

1 Introduction

The well-known LLL lattice reduction algorithm was presented in 1982 by Lenstra, Lenstra, Lovász [6]. Apart from various other applications (e.g. [12, Chapters 9, 10])

This is one of several papers published in *Designs, Codes and Cryptography* comprising the "Special Issue on Coding and Cryptography".

F. Fontein (✉)· U. Wagner
Universität Zürich, Zurich, Switzerland
e-mail: felix.fontein@math.uzh.ch

U. Wagner
e-mail: urs.wagner@math.uzh.ch

M. Schneider
Technische Universität Darmstadt, Darmstadt, Germany
e-mail: mischnei@cdc.informatik.tu-darmstadt.de

it has already at an early stage been used to attack various public key cryptosystems. Nevertheless lattice problems remain popular when it comes to the construction of provably secure cryptosystems (e.g. [9]). Consequently improvements in lattice reduction still have a direct impact on the security of many cryptosystems and rise high interest in the crypto-community.

Many lattice reduction algorithms used in practice are generalizations of the LLL algorithm. The Block–Korkine–Zolotarev (BKZ) reduction algorithm by Schnorr and Euchner [14] is probably the most used algorithm when stronger reduction than the one achieved by LLL is required. It can be seen as a generalization of LLL to higher blocksizes, and while the running time seems to behave well for small blocksizes [4], no useful upper bound has been proven so far. If the BKZ algorithm is terminated early, polynomial runtime can be proven, while still reaching results of similar quality [5]. Another improvement of the LLL algorithm has also been suggested in [14]. While in LLL adjacent basis vectors are swapped if certain conditions are satisfied, in the so called LLL with deep insertions (DeepLLL in the sequel), basis vectors can be swapped even when not adjacent. The practical behavior of DeepLLL when it comes to the reducedness of the output basis is superior the one of LLL. Unfortunately also the running time explodes and does not seem to be polynomial in the dimension of the lattice. One attempt to get across this problem is to restrict the insertions to certain blocks of basis vectors. While the authors in [14] claim that these blockwise restriction variants of DeepLLL run in polynomial time, we are not aware of any proof thereof. A different approach to produce DeepLLL reduced bases is taken in [2]; unfortunately, no proven bound on the running time is provided in that paper. Their approach does not fit into our framework of LLL-type algorithms.

For an overview on the practical behavior of the different variants and improvements on LLL, we refer to [4,11]. There the practical behavior of the reduction algorithms is investigated using the widely used fplll and NTL libraries, respectively.

In this paper we present two new versions of DeepLLL, called PotLLL and PotLLL2. To our knowledge it is the first improvement of LLL with regard to deep insertions which provably runs in polynomial time. The practical behavior of PotLLL and PotLLL2 regarding both the output quality and running time is empirically tested and compared to BKZ and DeepLLL with different blocksizes. The tests are performed with a completely new implementation of the different reduction algorithms. This additionally allows an independent review of the results in [4,11]. The tests indicate that our algorithm can serve as a serious alternative to BKZ with low blocksizes. This paper is an extension of the work presented at WCC 2013 in Bergen [3].

The paper is organized as follows. In Sect. 2 all necessary notations and definitions are given. In Sect. 3 the reduction notion and the new algorithm is presented and a theoretical analysis is provided. Sect. 4 contains the empirical results and conclusions are drawn in Sect. 5.

2 Preliminaries

A lattice $\mathcal{L} \subset \mathbb{R}^m$ of rank n and dimension m is a discrete subgroup of $\mathbb{R}^m$ generated by integer linear combinations of n linearly independent vectors $b_1, \ldots, b_n$ in $\mathbb{R}^m$:

$$\mathcal{L} = \mathcal{L}(b_1, \ldots, b_n) := \left\{ \sum_{i=1}^{n} x_i b_i \;\middle|\; \forall i : x_i \in \mathbb{Z} \right\}.$$

We will often write the basis $b_1, \ldots, b_n$ as rows of a matrix B in the following way $B = [b_1, \ldots, b_n]$. In order to have exact representations in computers, unless explicitly stated otherwise, only lattices in $\mathbb{Q}^n$ are considered. Simple scaling by the least common multiple of the denominators allows us to restrict ourselves to integer lattices $\mathcal{L} \subseteq \mathbb{Z}^m$. The volume of a lattice $\mathcal{L}(B)$ equals the volume of its fundamental parallelepiped $\mathrm{vol}(\mathcal{L}) = \sqrt{\det(BB^t)}$. For $n \geq 2$, a lattice has infinitely many bases as $\mathcal{L}(B) = \mathcal{L}(B')$ if and only if $\exists U \in GL_n(\mathbb{Z}) : B = UB'$. Therefore, the volume of a lattice is well defined. By $\pi_k : \mathbb{R}^m \to \mathrm{span}\{b_1, \ldots, b_{k-1}\}^{\perp}$ we denote the orthogonal projection from $\mathbb{R}^m$ onto the orthogonal complement of $\mathrm{span}\{b_1, \ldots, b_{k-1}\}$. In particular, $\pi_1 = \mathrm{id}_{\mathbb{R}^m}$ and $b_i^* := \pi_i(b_i)$ equals the i-th basis vector of the Gram–Schmidt orthogonalization $B^* = [b_1^*, \ldots, b_n^*]$ of B. By $\mu_{i,j} := \langle b_i, b_j^* \rangle / \langle b_j^*, b_j^* \rangle$, $j < i$, we denote the Gram–Schmidt coefficients. The Gram–Schmidt vectors can iteratively be computed by $\pi_i(b_i) = b_i^* = b_i - \sum_{j=1}^{i-1} \mu_{i,j} b_j^*$.

Throughout this paper, by $\|\cdot\|$ we denote the Euclidean norm and by $\lambda_1(\mathcal{L})$ we denote the length of a shortest non-zero vector in $\mathcal{L}$ with respect to the Euclidean norm: $\lambda_1(\mathcal{L}) := \min_{v \in \mathcal{L} \setminus \{0\}} \|v\|$. Determining $\lambda_1(\mathcal{L})$ is commonly known as the shortest vector problem (SVP) and is proven to be NP-hard (under randomized reductions) (see e.g. [8]). Upper bounds with respect to the determinant exist, for all rank n lattices $\mathcal{L}$ we have [7, Chapter 2]

$$\frac{\lambda_1(\mathcal{L})^2}{\mathrm{vol}(\mathcal{L})^{2/n}} \leq \gamma_n \leq 1 + \frac{n}{4},$$

where γ_n is the *Hermite constant* in dimension n. Given a relatively short vector $v \in \mathcal{L}$, one measures its quality by the *Hermite factor* $\|v\|/\mathrm{vol}(\mathcal{L})^{1/n}$ it achieves. Modern lattice reduction algorithms achieve a Hermite factor which is exponential in n and no polynomial time algorithm is known to achieve linear or polynomial Hermite factors.

Let S_n denote the group of permutations of n elements. By applying $\sigma \in S_n$ to a basis $B = [b_1, \ldots, b_n]$, the basis vectors are reordered $\sigma B = [b_{\sigma(1)}, \ldots, b_{\sigma(n)}]$. For $1 \leq k \leq \ell \leq n$ we define a class of elements $\sigma_{k,\ell} \in S_n$ as follows:

$$\sigma_{k,\ell}(i) = \begin{cases} i & \text{for } i < k \text{ or } i > \ell, \\ \ell & \text{for } i = k, \\ i-1 & \text{for } k < i \leq \ell. \end{cases} \tag{2.1}$$

Note that $\sigma_{k,\ell} = \sigma_{k,k+1}\sigma_{k+1,k+2}\ldots\sigma_{\ell-1,\ell}$ and that $\sigma_{k,k+1}$ is swapping the two elements k and $k+1$.

Definition 1 Let $\delta \in (1/4, 1]$. A basis $B = [b_1, \ldots, b_n]$ of a lattice $\mathcal{L}(b_1, \ldots, b_n)$ is called *δ-LLL reduced* if and only if it satisfies the following two conditions:

1. $\forall 1 \leq j < i \leq n : |\mu_{i,j}| \leq \frac{1}{2}$ (size-reduced).
2. $1 \leq k < n : \delta \cdot \|\pi_k(b_k)\|^2 \leq \|\pi_k(b_{k+1})\|^2$ (Lovász-condition).

A δ-LLL reduced basis $B = [b_1, \ldots, b_n]$ can be computed in polynomial time [6] and provably satisfies the following bounds:

$$\|b_1\| \leq \left(\delta - 1/4\right)^{-(n-1)/2} \cdot \lambda_1(\mathcal{L}(B)) \quad \text{and} \quad \|b_1\| \leq \left(\delta - 1/4\right)^{-(n-1)/4} \cdot \mathrm{vol}(\mathcal{L}(B))^{1/n}. \tag{2.2}$$

While these bounds can be reached, they are worst case bounds. In practice, LLL reduction algorithms behave much better [11]. One early attempt to improve the LLL reduction algorithm is due to Schnorr and Euchner [14] who came up with the notion of a DeepLLL reduced basis:

Definition 2 Let $\delta \in (1/4, 1]$. A basis $B = [b_1, \ldots, b_n]$ of a lattice $\mathcal{L}(b_1, \ldots, b_n)$ is called δ-DeepLLL *reduced with blocksize* β if and only if it satisfies the following two conditions:

1. $\forall 1 \leq j < i \leq n : |\mu_{i,j}| \leq \frac{1}{2}$ (size-reduced).
2. $\forall 1 \leq k < \ell \leq n$ with $k \leq \beta \vee \ell - k \leq \beta : \delta \cdot \|\pi_k(b_k)\|^2 \leq \|\pi_k(b_\ell)\|^2$.

If $\beta = n$ we simply call this a DeepLLL reduced basis. While the first basis vector of DeepLLL reduced bases in the worst case does not achieve a better Hermite factor than classical LLL (see Sect. 3.4), the according reduction algorithms usually return much shorter vectors than pure LLL. Unfortunately no polynomial time algorithm to compute DeepLLL reduced bases is known.

The following definition is used in the proof (see e.g. [8]) of the polynomial running time of the LLL reduction algorithm and will play a main role in our improved variant of LLL.

Definition 3 The *potential* $\mathrm{Pot}(B)$ of a lattice basis $B = [b_1, \ldots, b_n]$ is defined as

$$\mathrm{Pot}(B) := \prod_{i=1}^{n} \mathrm{vol}\left(\mathcal{L}(b_1, \ldots, b_i)\right)^2 = \prod_{i=1}^{n} \|b_i^*\|^{2(n-i+1)}.$$

Here it is used that $\mathrm{vol}(\mathcal{L}) = \prod_{i=1}^{n} \|b_i^*\|$. Note that, unlike the volume of the lattice, the potential of a basis is variant under basis permutations. The following lemma describes how the potential changes if $\sigma_{k,\ell}$ is applied to the basis.

Lemma 1 *Let $B = [b_1, \ldots, b_n]$ be a lattice basis. Then for $1 \leq k \leq \ell \leq n$*

$$\mathrm{Pot}(\sigma_{k,\ell} B) = \mathrm{Pot}(B) \cdot \prod_{i=k}^{\ell} \frac{\|\pi_i(b_\ell)\|^2}{\|\pi_i(b_i)\|^2}.$$

Proof First note that it is well-known that $\mathrm{Pot}(\sigma_{k,k+1} B) = \|\pi_k(b_{k+1})\|^2 / \|\pi_k(b_k)\|^2 \cdot \mathrm{Pot}(B)$. This property is used in the proofs of the polynomial running time of LLL [6,8].

We prove the claim by induction over $\ell - k$. The claim is true for $k = \ell$. For $k < \ell$, $\sigma_{k,\ell} = \sigma_{k,k+1}\sigma_{k+1,\ell}$. As b_ℓ is the $(k+1)$-th basis vector of $\sigma_{k+1,\ell} B$, with the above identity we get $\mathrm{Pot}(\sigma_{k,\ell} B) = \mathrm{Pot}(\sigma_{k,k+1}\sigma_{k+1,\ell} B) = \frac{\|\pi_k(b_\ell)\|^2}{\|\pi_k(b_k)\|^2} \cdot \mathrm{Pot}(\sigma_{k+1,\ell} B)$, which completes the proof. □

3 The Potential-LLL reduction

In this section we present our polynomial time variant of DeepLLL. We start with the definition of a δ-PotLLL reduced basis. Then we present an algorithm that outputs such a basis followed by a runtime proof.

Definition 4 Let $\delta \in (1/4, 1]$. A lattice basis $B = [b_1, \ldots, b_n]$ is δ-PotLLL *reduced* if and only if

1. $\forall 1 \leq j < i \leq n : |\mu_{i,j}| \leq \frac{1}{2}$ (size-reduced).
2. $\forall 1 \leq k < \ell \leq n : \delta \cdot \mathrm{Pot}(B) \leq \mathrm{Pot}(\sigma_{k,\ell} B)$.

Lemma 2 *A δ-PotLLL reduced basis B is also δ-LLL reduced.*

Proof Lemma 1 shows that $\delta \cdot \mathrm{Pot}(B) \leq \mathrm{Pot}(\sigma_{i,i+1} B)$ if and only if $\delta \|\pi_i(b_i)\|^2 \leq \|\pi_i(b_{i+1})\|^2$. Thus the Lovász condition is implied by the second condition in Definition 4 restricted to consecutive pairs, i.e. $\ell = k + 1$. □

Lemma 3 *For $\delta \in \left(4^{-1/(n-1)}, 1\right]$, a δ-DeepLLL reduced basis B is also δ^{n-1}-PotLLL reduced.*

Proof We proceed by contradiction. Assume that B is not δ^{n-1}-PotLLL reduced, i.e. there exist $1 \leq k < \ell \leq n$ such that $\delta^{n-1}\mathrm{Pot}(B) > \mathrm{Pot}(\sigma_{k,\ell}B)$. By Lemma 1 this is equivalent to

$$\delta^{n-1} > \prod_{i=k}^{\ell} \frac{\|\pi_i(b_\ell)\|^2}{\|\pi_i(b_i)\|^2} = \prod_{i=k}^{\ell-1} \frac{\|\pi_i(b_\ell)\|^2}{\|\pi_i(b_i)\|^2}.$$

It follows that there exist a $j \in [k, \ell-1]$ such that $\|\pi_j(b_\ell)\|^2/\|\pi_j(b_j)\|^2 < \delta^{(n-1)/(\ell-k)} \leq \delta$ which implies that B is not δ-DeepLLL reduced. □

3.1 High-level description

A high-level version of the algorithm is presented as Algorithm 1. The algorithm is very similar to the classical LLL algorithm and the classical DeepLLL reduction by Schnorr and Euchner [14]. During its execution, the first $\ell - 1$ basis vectors are always δ-PotLLL reduced (this guarantees termination of the algorithm). As opposed to classical LLL, and similar to DeepLLL, ℓ might decrease by more than one. This happens precisely during deep insertions: in these cases, the ℓ-th vector is not swapped with the $(\ell-1)$-th one, as in classical LLL, but is moved to the k-th position for $k < \ell - 1$. In case $k = \ell - 1$, this equals the swapping of adjacent basis vectors as in classical LLL. The main difference of PotLLL and DeepLLL is the condition that controls insertion of a vector: in DeepLLL, the designated insertion index k is chosen as $k = \min\left\{j \in \{1, \ldots, \ell\} \mid \|\pi_j(b_\ell)\|^2 \leq \delta \cdot \|\pi_j(b_j)\|^2\right\}$, while in PotLLL it is chosen as $k = \mathrm{argmin}_{1\leq j\leq \ell}\mathrm{Pot}(\sigma_{j,\ell}B)$.

Algorithm 1: Potential LLL

Input: Basis $B \in \mathbb{Z}^{n\times m}$, $\delta \in (1/4, 1]$
Output: A δ-PotLLL reduced basis.

```
1  δ-LLL reduce B
2  ℓ ← 1
3  while ℓ ≤ n do
4  |  Size-reduce(B)
5  |  k ← argmin_{1≤j≤ℓ} Pot(σ_{j,ℓ} B)
6  |  if δ · Pot(B) > Pot(σ_{k,ℓ} B) then
7  |  |  B ← σ_{k,ℓ} B
8  |  |  ℓ ← k
9  |  else
10 |  |  ℓ ← ℓ + 1
11 |  end
12 end
13 return B
```

3.1.1 Preprocessing

On line 1 we LLL reduce the input basis before proceeding. It turns out that while omitting this preprocessing does not change the output quality of the bases on average (see Fig. 5a), it is on average beneficial when it comes to the running time (see Fig. 5b). Note that most implementations of BKZ also preprocess the input basis with LLL and a more detailed discussion in Sect. 4.2.

3.1.2 PotLLL2

On line 5 the insertion depth is chosen such that the potential of the basis is minimal under the insertion. Alternatively one can choose the insertion place k as $\min\{k : \mathrm{Pot}(\sigma_{k,\ell}B) < \delta \cdot \mathrm{Pot}(B)\}$. Neither the running time analysis nor the fact that the output basis is PotLLL reduced is changed. We refer to this variant of PotLLL as PotLLL2.

3.2 Detailed description

There are two details to consider when implementing Algorithm 1. The first one is that since the basis vectors $b_1, \ldots, b_{\ell-1}$ are already δ-PotLLL reduced, they are in particular also size-reduced. Moreover, the basis vectors $b_{\ell+1}, \ldots, b_n$ will be considered later again. So in line 4 of the algorithm it suffices to size-reduce b_ℓ by $b_1, \ldots, b_{\ell-1}$ as in classical LLL. Upon termination, when $\ell = n + 1$, the whole basis will be size-reduced.

Another thing to consider is the computation of the potentials of B and $\sigma_{j,\ell}B$ for $1 \le j \le \ell$ in lines 5 and 6. Computing the potential of the basis is a rather slow operation. But we do not need to compute the potential itself, but only compare $\mathrm{Pot}(\sigma_{k,\ell}B)$ to $\mathrm{Pot}(B)$; by Lemma 1, this quotient can be efficiently computed. Define $P_{k,\ell} := \mathrm{Pot}(\sigma_{k,\ell}B)/\mathrm{Pot}(B)$. The "if"-condition in line 6 will then change to $\delta > P_{k,\ell}$, and the minimum in line 5 will change to $\mathrm{argmin}_{1\le j\le \ell} P_{j,\ell}$. Using $P_{\ell,\ell} = 1$ and

$$P_{j,\ell} = \frac{\mathrm{Pot}(\sigma_{j,\ell}B)}{\mathrm{Pot}(B)} = P_{j+1,\ell} \cdot \frac{\|\pi_j(b_\ell)\|^2}{\|\pi_j(b_j)\|^2} = P_{j+1,\ell} \cdot \frac{\|b_\ell^*\|^2 + \sum_{i=j}^{\ell-1} \mu_{\ell,i}^2 \|b_i^*\|^2}{\|b_j^*\|^2} \tag{3.1}$$

for $j < \ell$ (Lemma 1), we can quickly determine $\mathrm{argmin}_{1\le j\le \ell} P_{j,\ell}$ and check whether $\delta > P_{k,\ell}$ if j minimizes $P_{j,\ell}$.

A detailed version of Algorithm 1 with these steps filled in is described as Algorithm 2. On line 8 of Algorithm 2, $P_{j,\ell}$ is iteratively computed as in Eq. (3.1). Clearly, the algorithm could be further improved by iteratively computing $\|\pi_j(b_\ell)\|^2$ from $\|\pi_{j+1}(b_\ell)\|^2$. Depending on the implementation of the Gram–Schmidt orthogonalization, this might already have been computed and stored. For example, when using the Gram–Schmidt orthogonalization as described in Fig. 4 of [10], then $\|\pi_j(b_\ell)\|^2 = s_{j-1}$ after computation of $\|b_\ell^*\|^2$ and $\mu_{\ell,j}$ for $1 \le j < \ell$.

3.3 Complexity analysis

Here we show that the number of operations in the PotLLL algorithm is bounded polynomially in the dimension n and the logarithm of the input size. We present the runtime for Algorithm 2.

Proposition 1 *Let $\delta \in (1/4, 1)$ and $C = \max_{i=1\ldots n}\|b_i\|^2$. Then Algorithm 2 performs $\mathcal{O}(n^3 \log_{1/\delta}(C))$ iterations of the `while` loop in line 3 and a total of $\mathcal{O}(mn^4 \log_{1/\delta}(C))$ arithmetic operations.*

Proof Let us start by bounding from above the potential I of the input basis with respect to C. Let $d_j := \mathrm{vol}\left(\mathcal{L}(b_1, \ldots, b_j)\right)^2 = \prod_{i=1}^{j}\|b_i^*\|^2$ for $j = 1, \ldots, n$. Recall that $\|b_i^*\|^2 \le \|b_i\|^2 \le C$ for $i = 1, \ldots, n$ and hence $d_j < C^j$. Consequently we have the following upper bound on the potential

$$I = \prod_{j=1}^{n-1} d_j \cdot \mathrm{vol}\,(\mathcal{L})^2 \le \prod_{j=1}^{n-1} C^j \cdot \mathrm{vol}\,(\mathcal{L})^2 \le C^{\frac{n(n-1)}{2}} \cdot \mathrm{vol}\,(\mathcal{L})^2\,. \tag{3.2}$$

Algorithm 2: Potential LLL, detailed version

Input: Basis $B \in \mathbb{Z}^{n\times m}$, $\delta \in (1/4, 1]$
Output: A δ-PotLLL reduced basis.

1 δ-LLL reduce B
2 $\ell \leftarrow 1$
3 **while** $\ell \leq n$ **do**
4 Size-reduce(b_ℓ by $b_1, \ldots, b_{\ell-1}$)
5 Update($\|b_\ell^*\|^2$ and $\mu_{\ell,j}$ for $1 \leq j < \ell$)
6 $P \leftarrow 1, \quad P_{\min} \leftarrow 1, \quad k \leftarrow 1$
7 **for** $j = \ell - 1$ *down to* 1 **do**
8 $P \leftarrow P \cdot \frac{\|b_\ell^*\|^2 + \sum_{i=j}^{\ell-1} \mu_{\ell,i}^2 \|b_i^*\|^2}{\|b_j^*\|^2}$
9 **if** $P < P_{\min}$ **then**
10 $k \leftarrow j$
11 $P_{\min} \leftarrow P$
12 **end**
13 **end**
14 **if** $\delta > P_{\min}$ **then**
15 $B \leftarrow \sigma_{k,\ell} B$
16 Update($\|b_k^*\|^2$ and $\mu_{k,j}$ for $1 \leq j < k$)
17 $\ell \leftarrow k$
18 **else**
19 $\ell \leftarrow \ell + 1$
20 **end**
21 **end**
22 **return** B

Note that the first equality also shows that the potential of any basis of an integer lattice $\mathcal{L}$ is bounded from below by $\mathrm{vol}\,(\mathcal{L})^2$ as $d_j \geq 1$.

Now, by a standard argument, we show that the number of iterations of the while loop is bounded by $\mathcal{O}(n^3 \log_{1/\delta}(C))$. In each iteration, either the iteration counter ℓ is increased by 1, or an insertion takes place and ℓ is decreased by at most $n-1$. In the insertion case, the potential is decreased by a factor at least δ. So after N swaps the potential I_N satisfies $I \geq (1/\delta)^N I_N \geq (1/\delta)^N \cdot \mathrm{vol}\,(\mathcal{L})^2$ using $I_N \geq \mathrm{vol}\,(\mathcal{L})^2$. Consequently the number of swaps N is bounded by $N \leq \log_{1/\delta}(I/\mathrm{vol}\,(\mathcal{L})^2)$. By Eq. 3.2 we get that $N \leq \log_{1/\delta}\left(C^{n(n-1)/2}\right)$. Now note that the number M of iterations where ℓ is increased by 1 is at most $M \leq (n-1)\cdot N + n$. This shows that the number of iterations is bounded by $\mathcal{O}(n^3 \log_{1/\delta}(C))$.

Next we show that the number of operations performed in each iteration of the `while` loop is dominated by $\mathcal{O}(nm)$ operations. Size-reduction (line 4) and the first update step (line 5) can be done in $\mathcal{O}(nm)$ steps. The for-loop consists of $\mathcal{O}(n)$ iterations where the most expensive operation is the update of P in line 8. Therefore the loop requires $\mathcal{O}(nm)$ arithmetic operations. Insertion can be done in $\mathcal{O}(n)$ operations, whereas the second update in line 16 requires again $\mathcal{O}(nm)$ operations.

It follows that each iteration costs at most $\mathcal{O}(nm)$ arithmetic operations. This shows that in total the algorithm performs $\mathcal{O}(mn^4 \log(C))$ operations. □

3.4 Worst-case behavior

For $\delta = 1$, there exist so called *critical bases* which are δ-LLL reduced bases and whose Hermite factor reaches the worst case bound in (2.2) [15]. These bases can be adapted to form a DeepLLL reduced basis where the first vector reaches the worst case bound in (2.2).

Proposition 2 *For $\alpha = \sqrt{3/4}$, the rows of $B = A_n(\alpha)$ (see below) define a δ-DeepLLL reduced basis with $\delta = 1$ and $\|b_1\| = \frac{1}{(\delta - 1/4)^{(n-1)/4}} \mathrm{vol}\,(\mathcal{L}(A_n))^{1/n}$.*

$$A_n(\alpha) := \begin{pmatrix} 1 & 0 & \cdots & \cdots & \cdots & 0 \\ \frac{1}{2} & \alpha & \ddots & & & \vdots \\ \vdots & \frac{\alpha}{2} & \alpha^2 & \ddots & & \vdots \\ \vdots & \vdots & \frac{\alpha^2}{2} & \ddots & \ddots & \vdots \\ \vdots & \vdots & \vdots & \ddots & \alpha^{n-2} & 0 \\ \frac{1}{2} & \frac{\alpha}{2} & \frac{\alpha^2}{2} & \cdots & \frac{\alpha^{n-2}}{2} & \alpha^{n-1} \end{pmatrix} \tag{3.3}$$

It is worth mentioning that the lattice in Proposition 2 is defined over $\mathbb{R}$ and not over $\mathbb{Q}$. By approximating $\alpha = \sqrt{3/4}$ by rational numbers from above, we can find a sequence of lattices which are δ-DeepLLL reduced with $\delta = 1$ and satisfy $\|b_1\| \to \frac{1}{(3/4)^{(n-1)/4}} \mathrm{vol}\,(\mathcal{L}(A_n(\alpha)))^{1/n}$, which is the worst-case bound on LLL for $\delta = 1$.

Proof From the diagonal form of A_n it is easy to see that $\mathrm{vol}\,(\mathcal{L}) = \det(A_n) = \alpha^{n(n-1)/2}$. Hence $\|b_1\| = 1 = 1/\alpha^{(n-1)/2}\mathrm{vol}\,(\mathcal{L})$. It remains to show that A_n is DeepLLL reduced. Note that the orthogonalized basis B^* is a diagonal matrix with the same entries on the diagonal as B. Note that it is size reduced as for all $1 \le j < i \le n$ we have $\mu_{i,j} = \langle b_i, b_j^* \rangle / \langle b_j^*, b_j^* \rangle = \frac{1}{2}\alpha^{2(j-1)}/\alpha^{2(j-1)} = \frac{1}{2}$. Further, using that $\pi_j(b_i) = b_i^* + \sum_{\ell=j}^{i-1} \mu_{i,\ell} b_\ell^*$, we have that

$$\|\pi_j(b_i)\|^2 = \alpha^{2(i-1)} + \frac{1}{4}\sum_{\ell=j}^{i-1} \alpha^{2(\ell-1)} = \alpha^{2(j-1)} \left(\frac{1}{4} \sum_{\ell=0}^{i-j-1} \alpha^{2\ell} + \alpha^{2(i-j)} \right) .$$

As for $\alpha = \sqrt{3/4}$, we have that $\frac{1}{4}\sum_{\ell=0}^{i-j-1} \alpha^{2\ell} + \alpha^{2(i-j)} = 1$, and hence $\|\pi_j(b_i)\|^2 = \alpha^{2(j-1)} = \|\pi_j(b_j)\|^2$. Therefore, the norms of the projections for fixed j are all equal, and $A_n(\alpha)$ is δ-DeepLLL reduced with $\delta = 1$. □

Using Lemma 3, we obtain:

Corollary 1 *For $\alpha = \sqrt{3/4}$, the rows of $A_n(\alpha)$ define a δ-PotLLL reduced basis with $\delta = 1$ and $\|b_1\| = \frac{1}{(\delta - 1/4)^{(n-1)/4}} \mathrm{vol}\,(\mathcal{L}(A_n))^{1/n}$.*

4 Experimental results

Extensive experiments have been made to examine how the classical LLL reduction algorithm performs in practice [4,11]. We ran extensive experiments to compare our PotLLL algorithms to our implementations of LLL, DeepLLL, and BKZ.

4.1 Setting

We run the following algorithms, each with the standard reduction parameter $\delta = 0.99$:

1. classical LLL,
2. PotLLL and PotLLL2,

3. DeepLLL with blocksize $\beta = 5$ and $\beta = 10$,
4. BKZ with blocksize 5 (BKZ-5) and 10 (BKZ-10).

The implementations all use the same arithmetic back-end. Integer arithmetic is done using GMP, and Gram–Schmidt arithmetic is done as described in [10, Figs. 4 and 5]. As floating point types, `long double` (x64 extended precision format, 80 bit representation) and MPFR arbitrary precision floating point numbers are used with a precision as described in [10]. The implementations of DeepLLL and BKZ follow the classical description in [14]. PotLLL was implemented as described in Algorithm 2. Our implementation will be made publicly available.

We ran experiments in dimensions 40–400, considering the dimensions which are multiples of 10. Some algorithms become too slow in high dimensions, whence we restrict the dimensions for these as follows: For DeepLLL with $\beta = 10$ we ran experiments up to dimension 300 and for PotLLL2 and BKZ-10 up to dimension 350.

In each dimension, we considered 50 random lattices. More precisely, we used the HNF bases of the lattices of seed 0–49 from the SVP Challenge.[1]

All experiments were run on Intel® Xeon® X7550 CPUs at 2 GHz on a shared memory machine. For dimensions 40 up to 160, we used `long double` arithmetic, and for dimensions above 160, we used MPFR. In dimension 160, we did the experiments both using `long double` and MPFR arithmetic. The reduced lattices did not differ. In dimension 170, floating point errors prevented the `long double` arithmetic variant to complete on some of the lattices.

4.2 Preprocessing

As mentioned in Sect. 3.1, we added a "preprocessing" step to PotLLL, PotLLL2 and DeepLLL, by first running LLL without any deep insertions and with the same reduction parameter on the basis, and only then running PotLLL resp. DeepLLL. We performed all experiments both with and without this preprocessing, except that without preprocessing, we left out certain higher dimensions. More precisely, PotLLL was run until dimension 400, PotLLL2 was run until dimension 300, DeepLLL with $\beta = 5$ up to dimension 320, and DeepLLL with $\beta = 10$ up to dimension 250.

Figure 5a shows the average n-th root Hermite factor for the resulting bases. It appears that while preprocessing can have both a positive and negative impact on the output quality, it in general does not change the average n-th root Hermite factor. This was to be expected, since essentially we applied PotLLL resp. DeepLLL to two different bases of the same lattice: one in Hermite Normal Form, and the other 0.99-LLL reduced.

When comparing the timing results, on the other hand, there are large differences. Figure 5a shows the timing in dimensions 160 up to 400 for DeepLLL and PotLLL with and without preprocessing. The times for the algorithms with preprocessing include the time needed for applying LLL with $\alpha = 0.99$. It is clear that the algorithms with preprocessing are significantly faster than the ones without.

We conclude that while preprocessing does not change the output quality in average, it has a huge impact on the running time. For this reason, and also to have a better comparison to BKZ which always applies LLL first, we restricted to the algorithms with preprocessing for the rest of the experiments.

[1] http://www.latticechallenge.org/svp-challenge

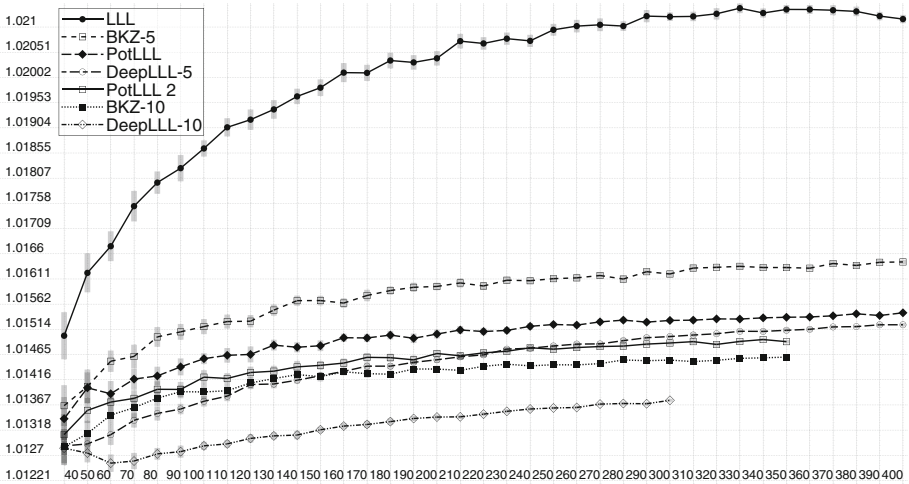

Fig. 1 Average n-th root Hermite factor (y axis) for dimension n (x axis) from 40 to 400

Table 1 Worst case bound and average case estimate for δ-LLL reduction, δ-DeepLLL reduction, δ-PotLLL reduction and δ-BKZ reduction of the n-th root Hermite factor $\|b_1\|^{1/n} \cdot \mathrm{vol}\,(\mathcal{L})^{-1/n^2}$

Dimension	$n = 100$	$n = 200$	$n = 300$	$n = 400$
Worst-case bound (proven)	≈ 1.0774	≈ 1.0778	≈ 1.0779	≈ 1.0780
Empirical 0.99-LLL	1.0186	1.0204	1.0212	1.0212
Empirical 0.99-BKZ-5	1.0152	1.0160	1.0162	1.0164
Empirical 0.99-PotLLL	1.0146	1.0151	1.0153	1.0155
Empirical 0.99-PotLLL2	1.0142	1.0147	1.0149	–
Empirical 0.99-DeepLLL with $\beta = 5$	1.0137	1.0146	1.0150	1.0152
Empirical 0.99-BKZ-10	1.0139	1.0144	1.0145	–
Empirical 0.99-DeepLLL with $\beta = 10$	1.0129	1.0134	1.0138	–

The entries are sorted in descending order with respect to the observed Hermite factors

4.3 Results

For each run, we recorded the length of the shortest vector as well as the required CPU time for the reduction. Our main interest lies in the n-th root of the *Hermite factor* $\frac{\|b_1\|}{\mathrm{vol}\,(\mathcal{L})^{1/n}}$, where b_1 is the shortest vector of the basis of $\mathcal{L}$ returned.

Figure 1 compares the average n-th root Hermite factor achieved by the different reduction algorithms in all dimensions. Also indicated are the confidence intervals for the average value with a confidence level[2] of 95 %. The average values for dimensions 100, 200, 300 and 400 are additionally summarized in Table 1, where also values of the worst-case bound from Eq. (2.2) are given. Note that our data for LLL is similar to the one in [11] and [4, Table 1]. However, we do not see convergence of the n-th root Hermite factors in our experiments, as they are

[2] To be able to compute confidence intervals, we assume that the data is distributed normally. We did some more experiments in dimensions 40, 50 and 60 to verify this conjecture: in case of the logarithm of the running time, this conjecture is quite accurate for most experiments; in case of the n-th root Hermite factor, it seems to be fine for most values, but there is some deviation at the tails.

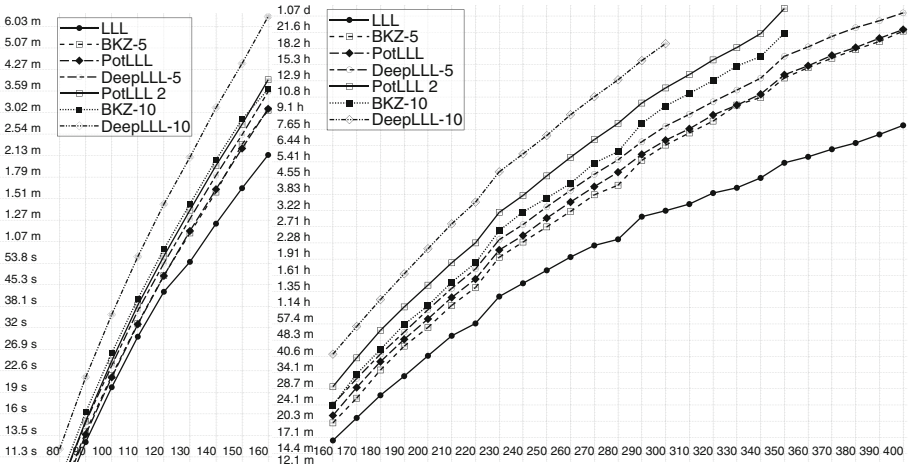

Fig. 2 Average logarithmic CPU time (y axis) for dimension n (x axis) from 40 to 400. The *left graph* uses `long double` arithmetic, the *right graph* MPFR arithmetic

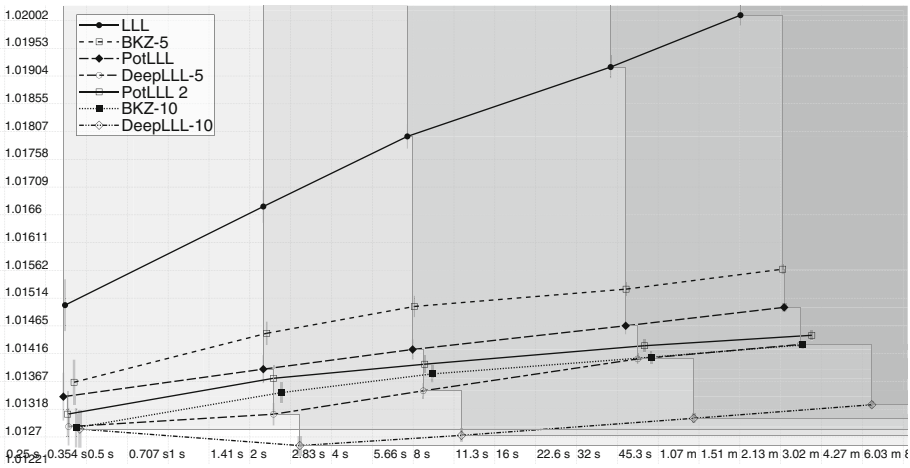

Fig. 3 `long double` arithmetic. The *highlighted areas* represent dimensions 40, 60, 80, 120 and 160

still increasing in high dimensions $n > 200$, respectively even slightly decreasing in the case of LLL. Our PotLLL algorithm clearly outperforms LLL and BKZ-5, however not PotLLL2, DeepLLL with $\beta = 5, 10$ and BKZ-10. DeepLLL with $\beta = 10$ seems the strongest of the considered lattice reduction algorithms. It is very interesting to see that PotLLL2 performs remarkably better than the original PotLLL when it comes to the Hermite factor achieved.

Figure 2 compares the average logarithmic running time of the algorithms for all dimensions. Recall that we used different arithmetic for dimensions below and above 160, whence two separate graphs are given. We see that the observed order is similar to the order induced by the Hermite factors. The only somewhat surprising fact is that PotLLL2 is even slower than BKZ-10, i.e. it is only faster than DeepLLL with $\beta = 10$.

Figures 3 and 4 allow to compare the different reduction algorithms with respect to the running time and the achieved Hermite factor at the same time. Every line connecting bullets

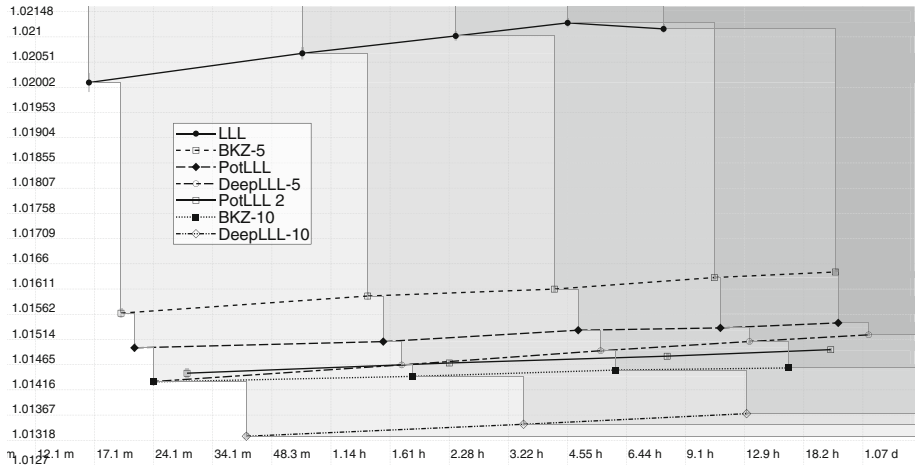

Fig. 4 MPFR arithmetic. The *highlighted areas* represent dimensions 160, 220, 280, 340 and 400

corresponds to the behavior of one algorithm in different dimensions. The gray box surrounding a bullet is the cartesian product of the two one-dimensional confidence intervals with confidence level 95 %. The shaded regions show which Hermite factors can be achieved in every dimension by these algorithms. Algorithms on the border of the region are optimal for their Hermite factor: none of the other algorithms in this list produces a better average Hermite factor in less time.

The only algorithm which is never optimal is PotLLL2, which is slower than DeepLLL with $\beta = 5$ or BKZ-10 and provides worse average Hermite factors up to dimension 160. PotLLL2 produces slightly better average Hermite factors than DeepLLL with $\beta = 5$ in high dimensions, for example from 280 on, but is there beaten by BKZ-10 which is in these dimensions far more efficient and provides better Hermite factors.

Another interesting observation is that in dimensions 40–80, PotLLL is both faster than BKZ-5 and yields shorter vectors. While the running time difference in dimension 80 is quite marginal, it is substantial in dimension 40. This shows that PotLLL could be used for efficient preprocessing of blocks for enumeration in BKZ-style algorithms with large block sizes, such as Chen's and Nguyen's BKZ 2.0 [1].

4.4 Comparison to fplll

To show the independence of the PotLLL concept from the concrete implementation, we added a PotLLL implementation to version 4.0.1 of the fplll library;[3] a patch can be downloaded at http://user.math.uzh.ch/fontein/fplll-potlll/. We ran the experiments with fplll's LLL implementation and our PotLLL addition in dimensions 40–320. For lower dimensions (up to 160 at least), the fplll-reduced lattices (both LLL and PotLLL) were identical to the ones of our implementation. For higher dimensions, the output quality in terms of the n-th root Hermite factor was essentially the same as for our implementation. While fplll was somewhat faster than our implementation, the relative difference between LLL and PotLLL was the same as for our implementation.

[3] http://perso.ens-lyon.fr/damien.stehle/fplll/

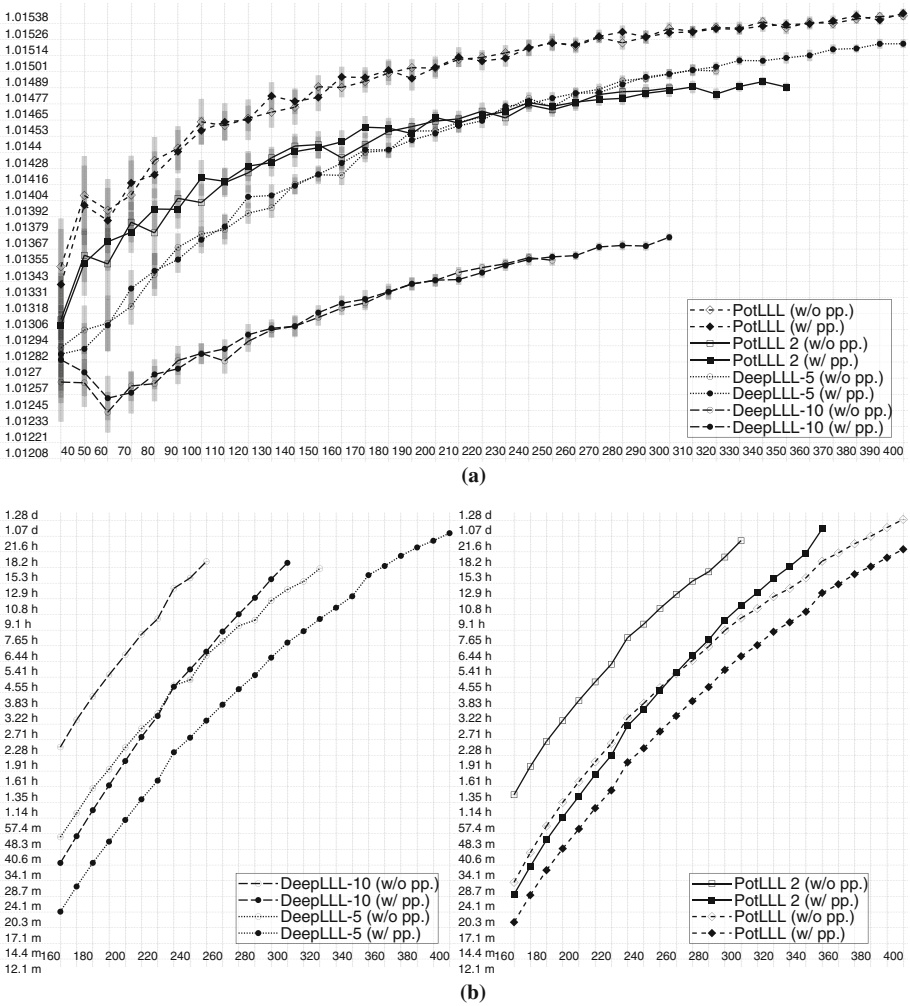

Fig. 5 Comparing PotLLL, PotLLL2 and DeepLLL variants with and without LLL preprocessing. **a** Comparison of the approximation factors with and without preprocessing. **b** Comparison of the running times with and without preprocessing (MPFR arithmetic)

5 Conclusion and future work

We define the notion of a PotLLL reduced basis and give two algorithms to compute such bases. Both algorithms are polynomial time improvements of LLL and are based on the concept of deep insertions as in Schnorr and Euchner's DeepLLL. While the provable bounds of the achieved Hermite factor are not better than for classical LLL—in fact, for reduction parameter $\delta = 1$, the existence of critical bases shows that better lattice-independent bounds do not exist—the practical behavior is much better than for classical LLL and they outperform BKZ-5. A comparison of the algorithms with respect to other quality measures, possibly including all basis vectors, is left for further work.

It is striking to see that although our two algorithms to compute a PotLLL reduced basis only differ in the strategy of choosing the insertion depth, their practical behavior is different.

We therefore believe that it might be worth to consider yet other strategies of choosing the insertions. Further an insertion can be seen as a special kind of permutation of the basis vectors. Ensuring that an insertion only happens when it results in a proper decrease of the potential of the basis ensures the polynomial running time of the algorithms. This concept could be generalized to other classes of permutations. The crucial point is the easy computation of the change of the potential under the different permutations.

It is likely that the improvements of the L^2 algorithm [11] and the $\tilde{L}^1$ algorithm [13] can be used to improve the runtime of our PotLLL algorithm, in order to achieve faster runtime. We leave this for future work.

Acknowledgments This work was supported by CASED (http://www.cased.de). Michael Schneider was supported by project BU 630/23-1 of the German Research Foundation (DFG). Urs Wagner and Felix Fontein are supported by SNF Grant no. 132256. The authors would like to thank the anonymous referees for their helpful comments. F. F. would also like to thank Kornelius Walter for the helpful discussions about statistics.

References

1. Chen Y., Nguyen P.Q.: BKZ 2.0: better lattice security estimates. In: Lee D.H., Wang X. (eds.) Advances in Cryptology—ASIACRYPT 2011. Lecture Notes in Computer Science, vol. 7073, pp. 1–20. Springer, Heidelberg (2011).
2. Cong L., Mow W.H., Howgrave-Graham N.: Reduced and fixed-complexity variants of the lll algorithm for communications. IEEE Trans. Commun. **61**(3), 1040–1050 (2013).
3. Fontein F., Schneider M., Wagner U.: A polynomial time version of LLL with deep insertions. In: Preproceedings of the International Workshop on Coding and Cryptography, WCC '13 (2013).
4. Gama N., Nguyen P.Q.: Predicting lattice reduction. In: Smart N. (ed.) Advances in Cryptology—EUROCRYPT 2008. LNCS, vol. 4965, pp. 31–51. Springer, Heidelberg (2008).
5. Hanrot G., Pujol X., Stehlé D.: Analyzing blockwise lattice algorithms using dynamical systems. In: Rogaway P. (ed.) CRYPTO. Lecture Notes in Computer Science, vol. 6841, pp. 447–464. Springer, Heidelberg (2011).
6. Lenstra A.K., Lenstra Jr H.W., Lovász L.: Factoring polynomials with rational coefficients. Math. Ann. **261**(4), 515–534 (1982).
7. Martinet J.: Perfect lattices in Euclidean Spaces. Grundlehren der Mathematischen Wissenschaften (Fundamental Principles of Mathematical Sciences), vol. 327. Springer-Verlag, Berlin (2003).
8. Micciancio D., Goldwasser S.: Complexity of Lattice Problems: A Cryptographic Perspective. The Kluwer International Series in Engineering and Computer Science, vol. 671. Kluwer Academic Publishers, Boston (2002).
9. Micciancio D., Regev O.: Lattice-based cryptography. In: Bernstein D.J., Buchmann J., Dahmen E. (eds.) Post-quantum Cryptography, pp. 147–191. Springer, Heidelberg (2008).
10. Nguyen P.Q., Stehlé D.: Floating-point LLL revisited. In: Cramer R. (ed.) Advances in Cryptology—EUROCRYPT 2005. LNCS, vol. 3494, pp. 215–233. Springer, Heidelberg (2005).
11. Nguyen P.Q., Stehlé D.: LLL on the average. In: Hess F., Pauli S., Pohst M.E. (eds.) ANTS. Lecture Notes in Computer Science, vol. 4076, pp. 238–256. Springer, Heidelberg (2006).
12. Nguyen P.Q., Vallée B.: The LLL Algorithm: Survey and Applications. Information Security and Cryptography. Springer, Heidelberg (2010).
13. Novocin A., Stehlé D., Villard G.: An LLL-reduction algorithm with quasi-linear time complexity: extended abstract. In: STOC, pp. 403–412. ACM, New York (2011).
14. Schnorr C.-P., Euchner M.: Lattice basis reduction: improved practical algorithms and solving subset sum problems. Math. Program. **66**(2), 181–199 (1994).
15. Schnorr C.-P.: Block reduced lattice bases and successive minima. Comb. Prob. Comput. **3**, 507–522 (1994).